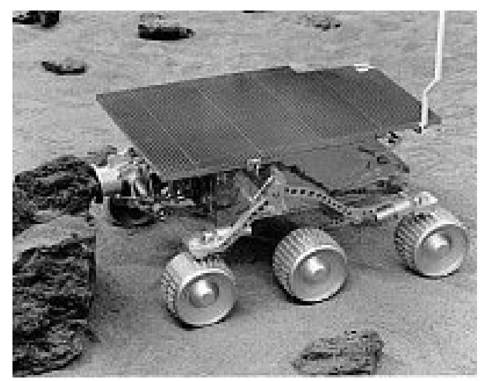
Vertically Integrated Analysis and Transformation for Embedded Software

John Regehr
University of Utah

Embedded Systems

- Most new microprocessors are embedded
  - Consumer electronics
  - Vehicle control systems
  - Medical equipment
  - Smart dust



Problem

- Compared to general-purpose compilation:
  - We have a lot more information
  - We have a lot more constraints
- So, using standard toolchain:
  - Ignores most of the information
  - Ignores most of the constraints
- However:
  - Strong economic incentive to avoid reinventing the wheel

Vertically Integrated Program Analysis (VIPA)

- Framework for combining analyses and transformations operating on a system at multiple levels of abstraction

(1) What good is gcc?

- Traditional C/C++ compilers pretty close to end of the line with respect to optimizing embedded SW
- However, C/C++ still useful for a while
- VIPA:
  - Keep the compiler around as a code generator
  - But do analysis and coarse-grain transformation in separate tools

(2) Tradeoffs are hard

- Embedded compilers must make difficult tradeoffs between goals
  - Power use, code size, data size, avoid crashing, etc…
  - Each embedded system has a different prioritization for these goals
- Standard compilers are ill-equipped to do what we want
  - Mechanism and policy all mixed up

(3) Levels of abstraction

- Analyses and transformations need to be performed at multiple levels of abstraction
  - Model – task mapping, exclusive modes, real-time deadlines
  - Source – concurrency, exceptions
  - Binary – memory usage, execution time, bit widths
- Standard compilers are ill-equipped to do what we want

(4) Tools are myopic

- Analysis tools often return binary results
  - “System is not schedulable”
  - “Network acceptor thread contains a potential stack overflow”
- Often more information is available but hidden
  - Which task is blocking schedulability?
  - What is the path to max stack depth?
- This information can be exploited

(5) Analysis good!

- Increasing asymmetry between
  - Resources on a PC and
  - Resources on a typical embedded system
- Program analysis and transformation tools are rapidly becoming more useful and effective
- The asymmetry can and should be exploited

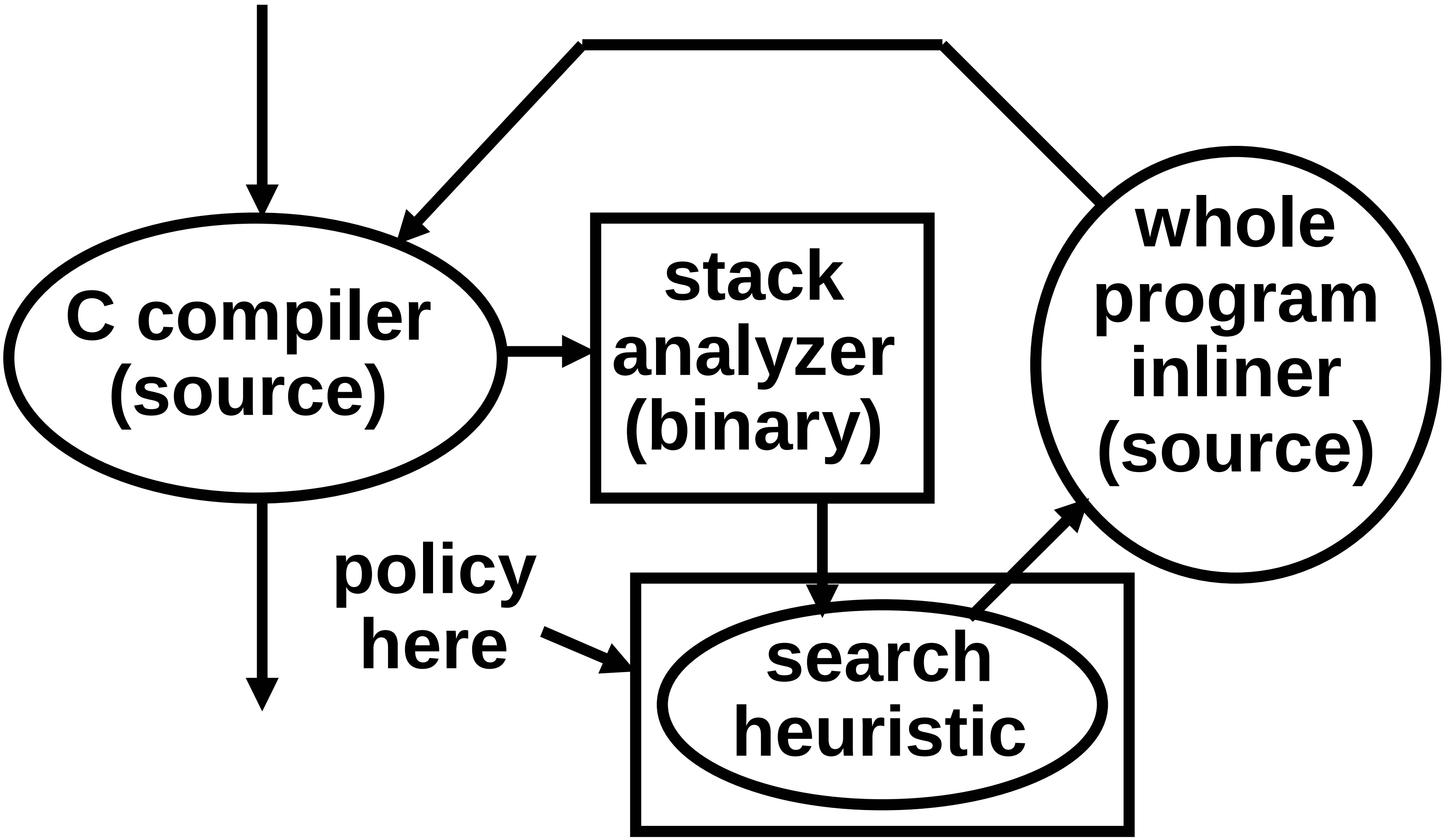
VIPA Example 1

- **Given:**
  - **Tool to compute a static upper bound on stack memory usage**
  - **Global function inlining tool**
- **Goal:**
  - **Reduce the stack memory requirements of an embedded system**

Reducing Stack Depth [EMSOFT 2003]

- **Observation: Function inlining often decreases stack requirements**
  - **Avoids pushing registers, frame pointer, return address**
  - **Called code can be specialized**
- **Strategy: Use stack tool output as input to global inlining tool**

Feedback Loop



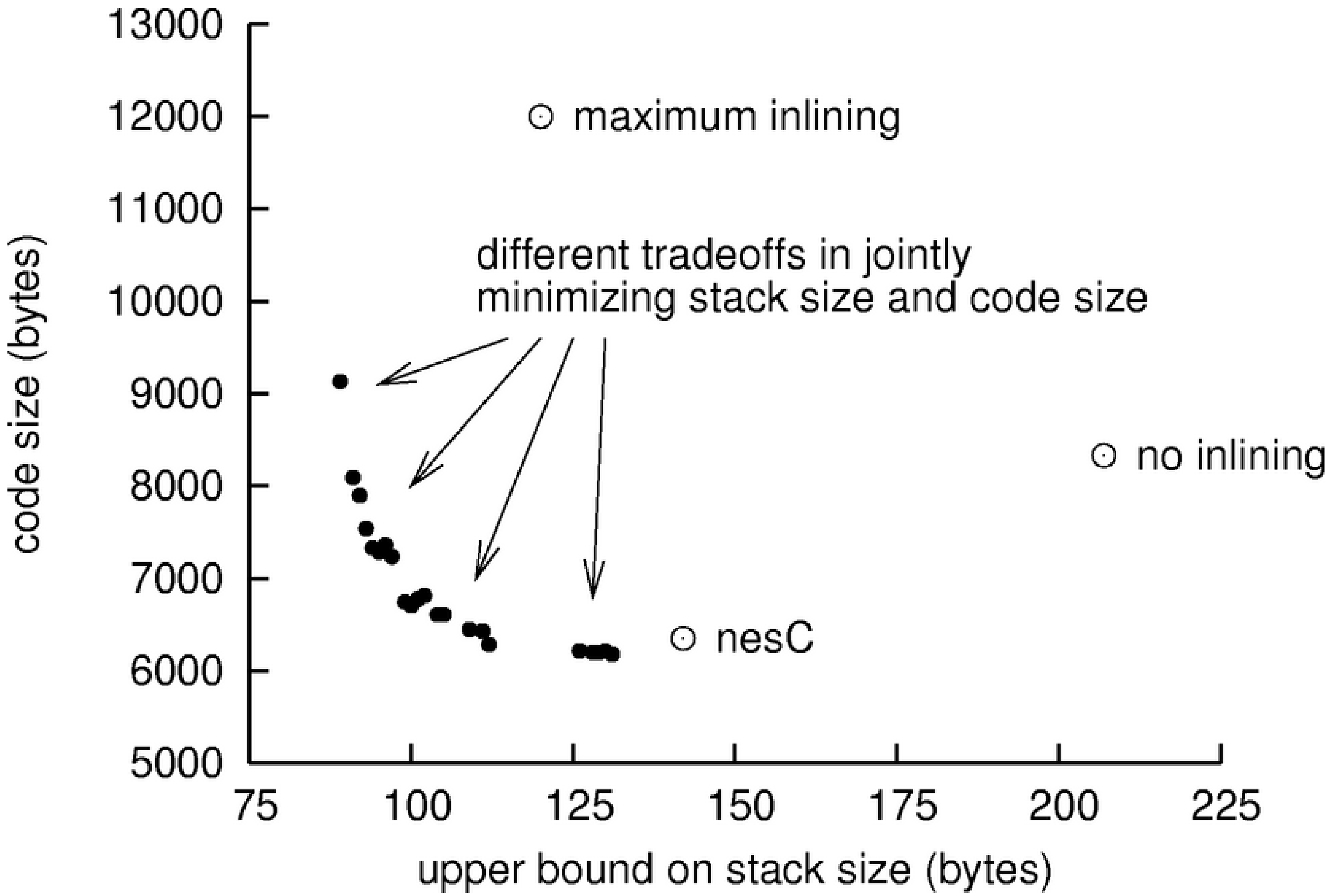
Challenges

1. **Inlining causes code bloat**
   - **Solution: Minimize user-defined cost function that balances stack memory and code size**
2. **Inlining sometimes increases stack depth!**
   - **Solution: Trial compilations**
3. **Search space is exponential in number of static function calls**
   - **Solution: Heuristic search**

Results

- Averaged over a bunch of TinyOS kernels…
  - 60% reduction in stack requirements compared to no inlining
  - 32% reduction compared to whole-program inlining not aimed at reducing stack depth

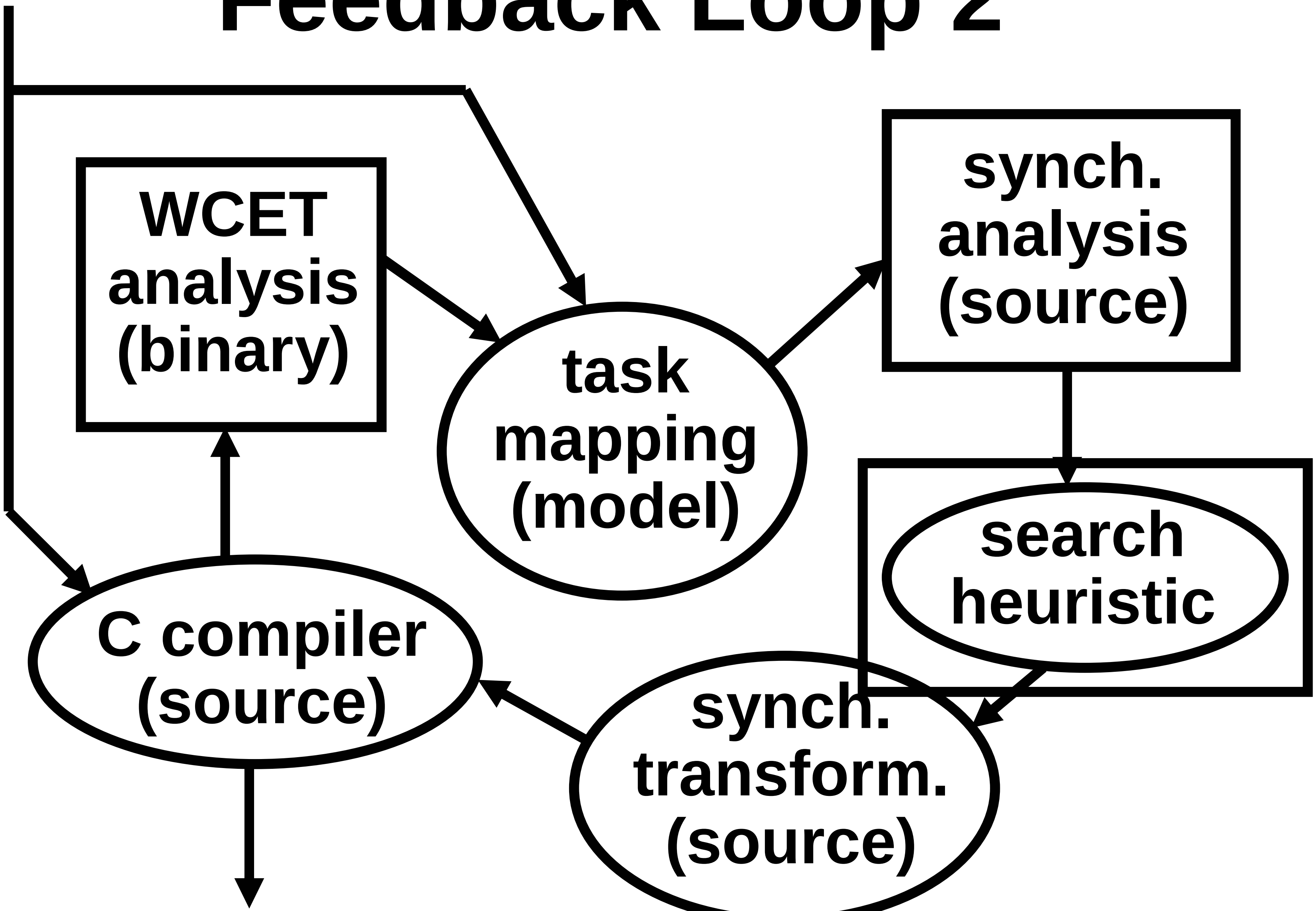
Result Details



VIPA Example 2

- Given:
  - WCET analysis
  - Synchronization analysis
    - Race / deadlock detection
  - Synchronization transformation
    - Lock elimination
    - Lock coarsening
  - Real-time-aware task mapping
- Goal: balance response time, efficiency, and memory use

Feedback Loop 2



VIPA Example N

- **Given:**
  - **Many, many tools that exist for analyzing and transforming embedded software**
- **Goal:**
  - **Rapidly produce efficient and reliable software**

What is VIPA?

- Exchange formats for analysis results
  - Annotated callgraph
  - Annotated task set
  - Others?
    - Type and alias information
    - Heap allocation / deallocation protocols
- Tools “opened up” to read/write the exchange formats

What is VIPA? Cont'd

- Strategies for connecting tools
  - E.g. feedback loops
- Policies
  - Manage tradeoffs between goals
- Auxiliary tools (that don't exist yet)
  - GUI to help developers specify tradeoffs
  - Manage interactions between analyses

Research Challenges

- Maintaining invariants
  - Transformations will invalidate some analysis results
- Avoiding bloat in the trusted computing base
  - Embedded developers have a hard time trusting even the compiler
- Avoiding long build times
- Providing good error messages

Related Work

- Phasing of optimizations inside compilers
- Model based design of embedded software
- MOBIES analysis interchange format

Conclusion

- **Benefits for developers:**
  - **Keep using the standard toolchain**
  - **Write straightforward code**
    - **Fewer fragile manual specializations**
  - **Explicit support for meeting design goals in the presence of tradeoffs**
    - **Policies externalized**
- **Benefit for researchers:**
  - **Lots of cool tools out there – let's make them play together**

More info here:

http://www.cs.utah.edu/~regehr/